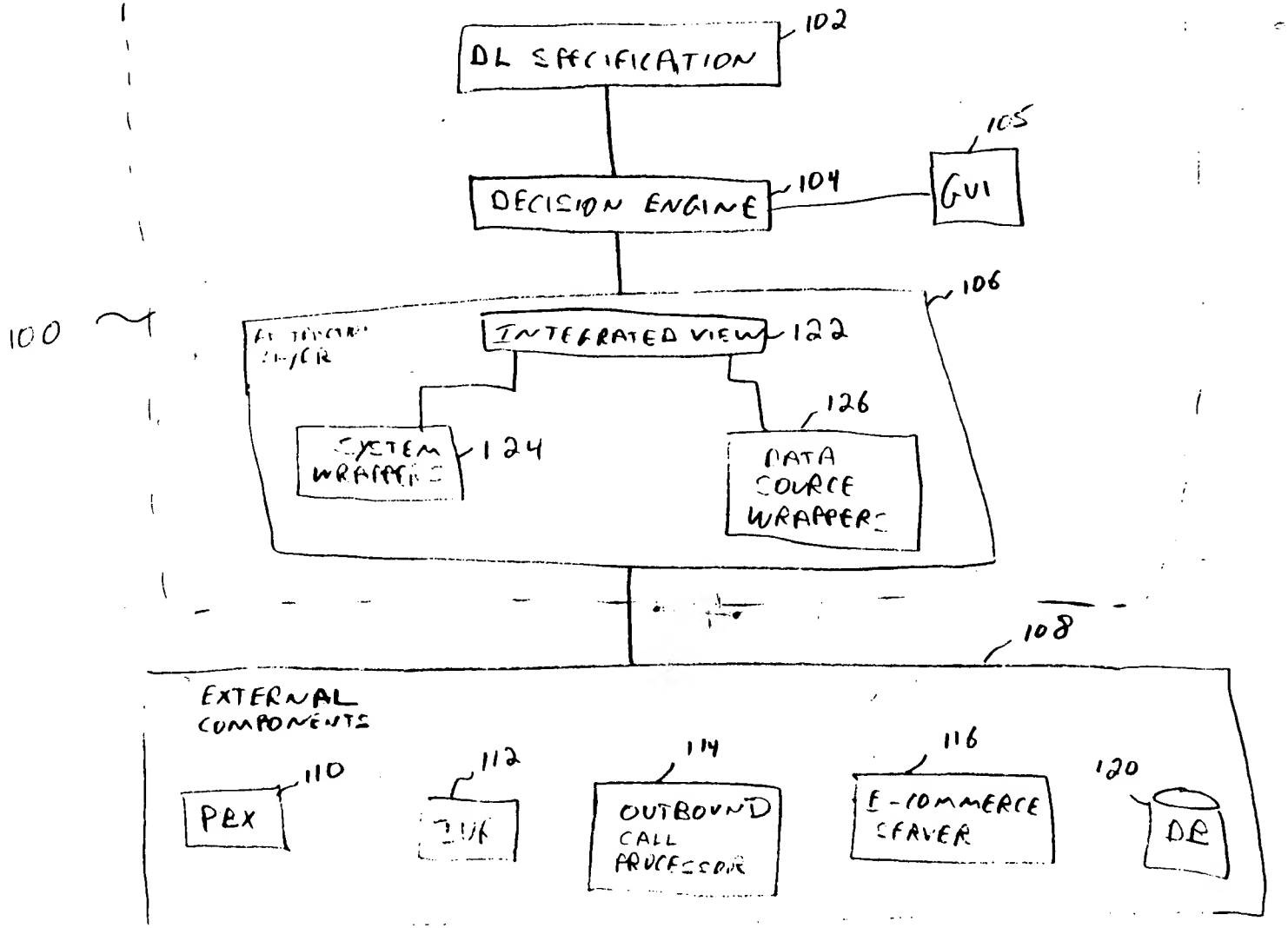
100
DL SPECIFICATION
102
DECISION ENGINE
104
GUI
105
INTEGRATED VIEW
122
106
SYSTEM WRAPPERS
124
DATA SOURCE WRAPPERS
126
108
EXTERNAL COMPONENTS
PBX
110
IVR
112
OUTBOUND CALL PROCESSOR
114
E-COMMERCE SERVER
116
DB
120

FIG 1

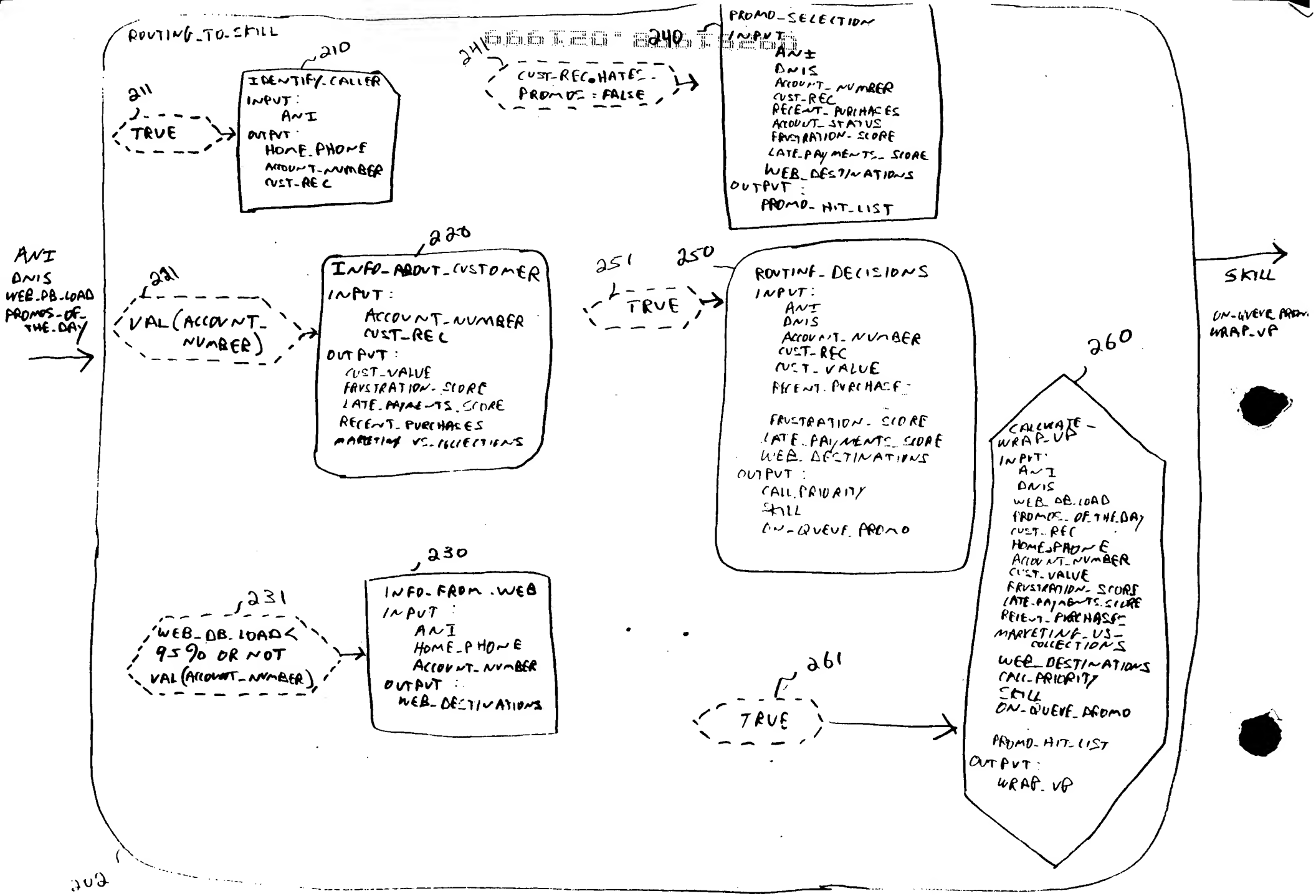
ROUTING_TO_SKILL
210
IDENTIFY_CALLER
INPUT:
ANI
OUTPUT:
HOME_PHONE
ACCOUNT_NUMBER
CUST_REC
211
TRUE
240
PROMO_SELECTION
INPUT:
ANI
DNIS
ACCOUNT_NUMBER
CUST_REC
RECENT_PURCHASES
ACCOUNT_STATUS
FRUSTRATION_SCORE
LATE_PAYMENTS_SCORE
WEB_DESTINATIONS
OUTPUT:
PROMO_HIT_LIST
241
CUST_REC.HATES_
PROMOS = FALSE
ANI
DNIS
WEB_DB_LOAD
PROMOS_OF_THE_DAY
220
INFO_ABOUT_CUSTOMER
INPUT:
ACCOUNT_NUMBER
CUST_REC
OUTPUT:
CUST_VALUE
FRUSTRATION_SCORE
LATE_PAYMENTS_SCORE
RECENT_PURCHASES
MARKETING_VS_COLLECTIONS
221
VAL(ACCOUNT_NUMBER)
250
ROUTING_DECISIONS
INPUT:
ANI
DNIS
ACCOUNT_NUMBER
CUST_REC
CUST_VALUE
RECENT_PURCHASES
FRUSTRATION_SCORE
LATE_PAYMENTS_SCORE
WEB_DESTINATIONS
OUTPUT:
CALL_PRIORITY
SKILL
ON_QUEUE_PROMO
251
TRUE
SKILL
ON-QUEUE PROMO
WRAP_UP
260
CALCULATE_WRAP_UP
INPUT:
ANI
DNIS
WEB_DB_LOAD
PROMOS_OF_THE_DAY
CUST_REC
HOME_PHONE
ACCOUNT_NUMBER
CUST_VALUE
FRUSTRATION_SCORE
LATE_PAYMENTS_SCORE
RECENT_PURCHASES
MARKETING_VS_COLLECTIONS
WEB_DESTINATIONS
CALL_PRIORITY
SKILL
ON_QUEUE_PROMO
PROMO_HIT_LIST
OUTPUT:
WRAP_UP
230
INFO_FROM_WEB
INPUT:
ANI
HOME_PHONE
ACCOUNT_NUMBER
OUTPUT:
WEB_DESTINATIONS
231
WEB_DB_LOAD < 95% OR NOT VAL(ACCOUNT_NUMBER)
261
TRUE
202

FIG. 2

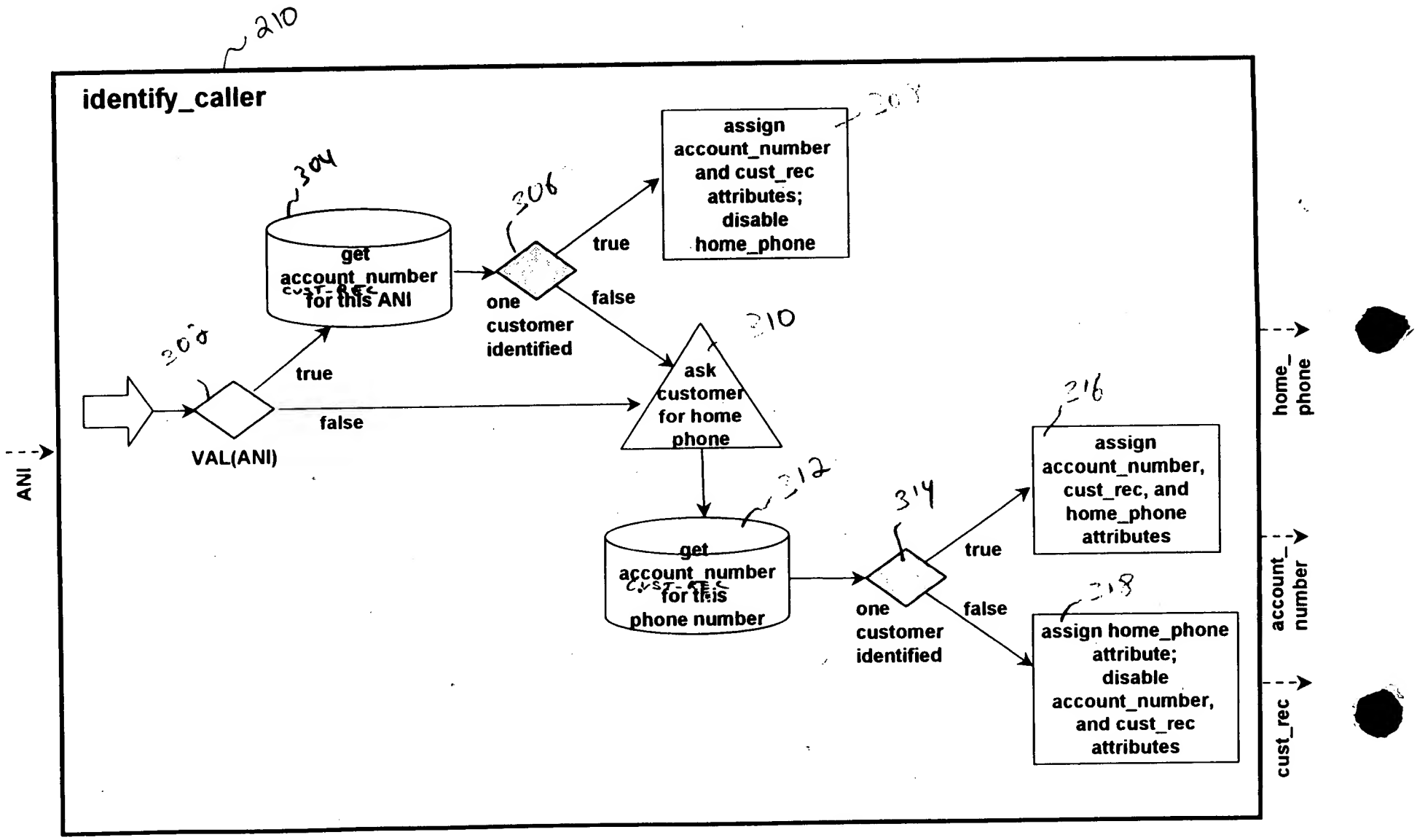
identify_caller
210
ANI
VAL(ANI)
200
true
false
get account_number cust_rec for this ANI
304
one customer identified
306
true
assign account_number and cust_rec attributes; disable home_phone
208
false
ask customer for home phone
310
get account_number cust_rec for this phone number
312
one customer identified
314
true
assign account_number, cust_rec, and home_phone attributes
216
false
assign home_phone attribute; disable account_number, and cust_rec attributes
218
home_phone
account_number
cust_rec

Fig. 3

```
Module:  identify_caller
   Submodule of:  routing_to_skill
   Input attributes:     ANI : 9digits
   Output attributes:    home_phone : 9digits
                         account_number : 15digits
                         cust_rec : tuple( name: string,
                                           address: string,
                                           card_color: {"platinum",
                                           "gold", "green"},
                                           hates_promos? : boolean,
                                           estimated_income_bracket :
                                           {"0-10K",">10K-20K",...,
                                           ">100K-150K", ">150K"},
                                           last_sent_bonus_check:date)
   Enabling condition:  true
   Type:                flowchart
   Computation:         See Fig. 3
   Side-effect:         yes
   Side Effect function: (IVR Dip)
```

FIG. 4

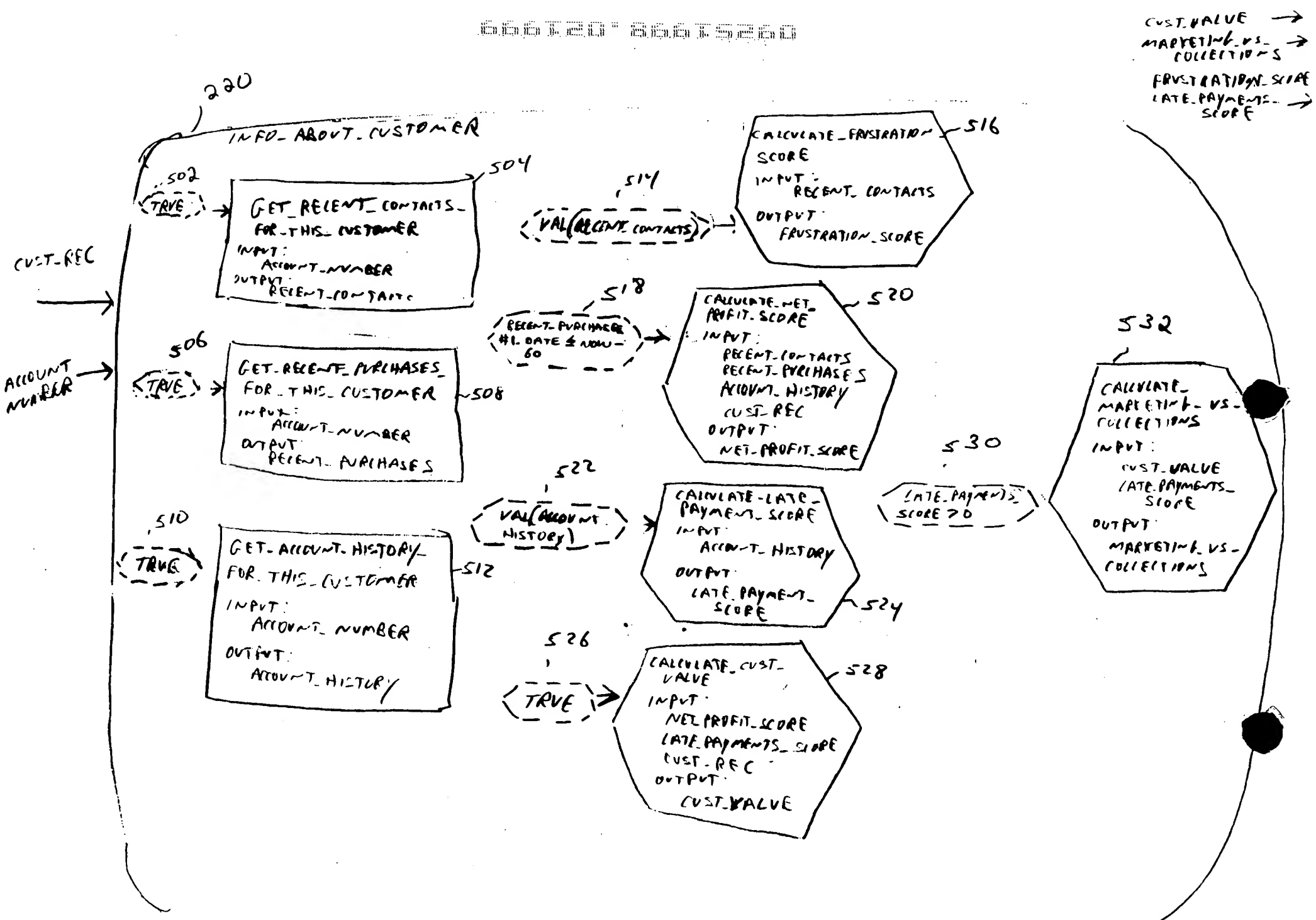
CUST_VALUE
MARKETING_VS_COLLECTIONS
FRUSTRATION_SCORE
LATE_PAYMENTS_SCORE
220
INFO_ABOUT_CUSTOMER
CUST_REC
ACCOUNT NUMBER
502
TRUE
504
GET_RECENT_CONTACTS_FOR_THIS_CUSTOMER
INPUT:
ACCOUNT_NUMBER
OUTPUT:
RECENT_CONTACTS
506
TRUE
508
GET_RECENT_PURCHASES_FOR_THIS_CUSTOMER
INPUT:
ACCOUNT_NUMBER
OUTPUT:
RECENT_PURCHASES
510
TRUE
512
GET_ACCOUNT_HISTORY_FOR_THIS_CUSTOMER
INPUT:
ACCOUNT_NUMBER
OUTPUT:
ACCOUNT_HISTORY
514
VAL(RECENT_CONTACTS)
516
CALCULATE_FRUSTRATION_SCORE
INPUT:
RECENT_CONTACTS
OUTPUT:
FRUSTRATION_SCORE
518
RECENT_PURCHASE #1. DATE ≤ NOW - 60
520
CALCULATE_NET_PROFIT_SCORE
INPUT:
RECENT_CONTACTS
RECENT_PURCHASES
ACCOUNT_HISTORY
CUST_REC
OUTPUT:
NET_PROFIT_SCORE
522
VAL(ACCOUNT HISTORY)
524
CALCULATE_LATE_PAYMENT_SCORE
INPUT:
ACCOUNT_HISTORY
OUTPUT:
LATE_PAYMENT_SCORE
526
TRUE
528
CALCULATE_CUST_VALUE
INPUT:
NET_PROFIT_SCORE
LATE_PAYMENTS_SCORE
CUST_REC
OUTPUT:
CUST_VALUE
530
LATE_PAYMENTS SCORE > 0
532
CALCULATE_MARKETING_VS_COLLECTIONS
INPUT:
CUST_VALUE
LATE_PAYMENTS_SCORE
OUTPUT:
MARKETING_VS_COLLECTIONS

FIG. 5

```
1   Module:  info_about_customer
2      Submodule of:  routing_to_skill
3      Input attributes:     account_number
4                            cust_rec
5      Output attributes:    cust_value : [1..10]
6                            frustration_score : [1..10]
7                            late_payments_score : [1..10]
8                            recent_purchases :list(tuple( date : date,
9                                                          item : 20digit,
10                                                         qty : int,
11                                                         amount: $value ))
12                           marketing_vs_collections : {"market",
13                           "collect"}
14
15  Enabling condition:      VAL(account_number)
16     Type:          declarative
17     Side-effect:   no
```

Fig. 6

```
1   Module: info_from_web

2       Submodule of:  routing_to_skill

3       Input attributes:     ANI
4                             home_phone
5                             account_number

6       Output attributes:    web_destinations : list(tuple(regions: set of
7                                                       {"Australia","Asia",...
8                                                       "NAmerica-US", "US"),
9                                                    itinerary:web_form_content,
10                                                   date_last_modified : date ))

11      Enabling condition:   web_db_load < 95% or not VAL(account_number)

12      Type:                 foreign

13      Computation:          get_web_data(ANI, home_phone, account_number)

14      Side-effect:          no
```

Fig. 7

```
1   Module:  promo_selection

2       Submodule of:  routing_to_skill

3       Input attributes:     ANI
4                             DNIS
5                             account_number
6                             cust_rec
7                             cust_value
8                             recent_purchases
9                             frustration_score
10                            late_payments_score
11                            web_destinations

12      Output attributes:    promo_hit_list : list ( promo_message )

13      Enabling condition:   cust_rec.hates_promos? = false

14      Type:                 foreign

15      Computation:          get_promo_hit_list(ANI, DNIS, account_number,
16                            cust_rec, cust_value, recent_purchases,
17                            account_status, frustration_score,
18                            late_payments_score, web_destinations)
19      Side-effect:          no
```

Fig. 8

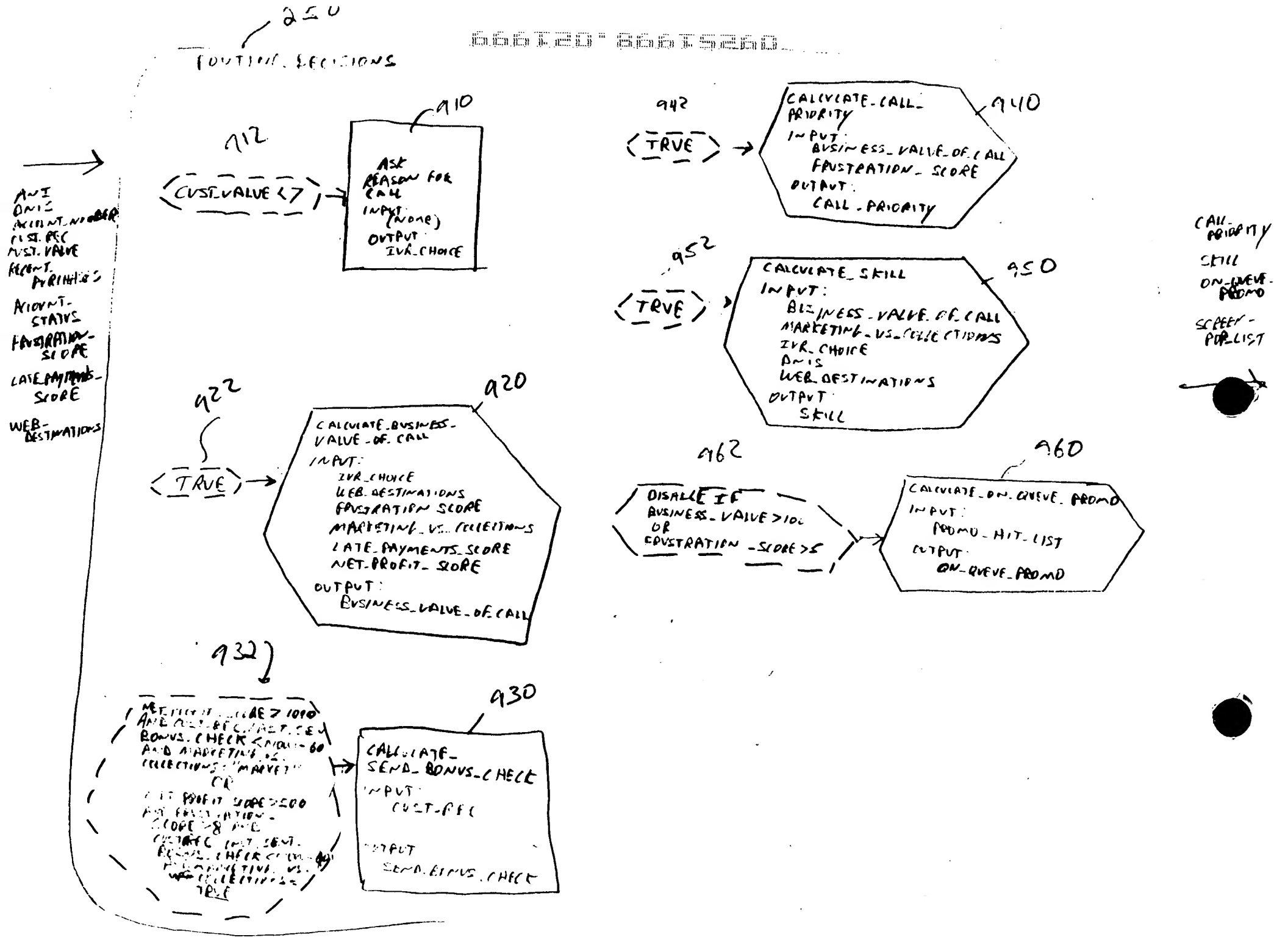
250
ROUTING_DECISIONS
910
912
CUST_VALUE < 7
ASK REASON FOR CALL
INPUT: (NONE)
OUTPUT: IVR_CHOICE
940
942
TRUE
CALCULATE_CALL_PRIORITY
INPUT: BUSINESS_VALUE_OF_CALL, FRUSTRATION_SCORE
OUTPUT: CALL_PRIORITY
950
952
TRUE
CALCULATE_SKILL
INPUT: BUSINESS_VALUE_OF_CALL, MARKETING_VS_COLLECTIONS, IVR_CHOICE, ANIS, WEB_DESTINATIONS
OUTPUT: SKILL
920
922
TRUE
CALCULATE_BUSINESS_VALUE_OF_CALL
INPUT: IVR_CHOICE, WEB_DESTINATIONS, FRUSTRATION SCORE, MARKETING_VS_COLLECTIONS, LATE_PAYMENTS_SCORE, NET_PROFIT_SCORE
OUTPUT: BUSINESS_VALUE_OF_CALL
960
962
DISABLE IF BUSINESS_VALUE > 10 OR FRUSTRATION_SCORE > 5
CALCULATE_ON_QUEUE_PROMO
INPUT: PROMO_HIT_LIST
OUTPUT: ON_QUEUE_PROMO
930
932
CALCULATE_SEND_BONUS_CHECK
INPUT: CUST_REC
OUTPUT: SEND_BONUS_CHECK
ANI
DNIS
ACCOUNT_NUMBER
CUST_REC
CUST_VALUE
RECENT_PURCHASES
ACCOUNT_STATUS
FRUSTRATION_SCORE
LATE_PAYMENTS_SCORE
WEB_DESTINATIONS
CALL_PRIORITY
SKILL
ON_QUEUE_PROMO
SCREEN_POP_LIST

FIG. 9

```
1   Module:  routing_decisions

2         Submodule of:      routing_to_skill

3         Input attributes:  ANI
4                            DNIS
5                            account_number
6                            cust_rec
7                            cust_value
8                            recent_purchases
9                            frustration_score
10                           late_payments_score
11                           web_destinations

12        Output attributes:call_priority : [1..4] \\corresponds to "low",
13                                 "med", "high", "top"
14                           skill : {"norm_tier_dom", "norm_tier_intl",
15                                  "australia_promo", "high_tier",
16                                       collections"}
17                           on_queue_promo : message_identifier
18                           screen_pop_list : list ( screen_entry )

19      Enabling condition:  true

20      Type:           declarative

21      Side-effect:         yes
```

Fig. 10

```
1   Module:  calculate_wrap_up

2       Submodule of:  routing_to_skill

3       Input attributes:     Ani
4                             dnis
5                             Web_DB_Load
6                             Promos_Of_The_Day
7                             Cust_Rec
8                             Home_Phone
9                             Account_Number
10                            Cust_Value
11                            Frustration_Score
12                            Late_Payments_Score
13                            Recent_Purchases
14                            Marketing_VS_Collections
15                            Web_Destinations
16                            Call_Priority
17                            Skill
18                            On_Queue_Promo
19                            Screen_Pop_List
20                            Promo_Hit_List

21      Output attributes:    wrap_up : set ( tuple ( att_name: string,
22                                                    value: string ))
23      Enabling condition:   true

24      Type:          decision

25      Computation:
26         Rules:             if true then wrap_up <- (att_name: "DNIS",
27                                    value : convert-to-string (DNIS))
28                            if true then wrap_up <- (att_name: "ANI",
29                                    value: convert-to-string (ANI))
30                            if true then wrap_up <- (att_name: "skill",
31                                    value: skill)
32                            if web_destinations not empty then wrap_up <-
33                                    (att_name: \"web_destinations",
34                                    value: (convert-to-string
35                                          (web_destinations))
36                            if cust_rec.card_color = "gold" <-
37                                   (att_name:"frustration_score",
38                                    value: convert-to-string
39                                    (frustration_score))
40          Combining policy:       wrap-up-cp //use contributions of all
41                                                 rules with true condition
42          Side-effect:            yes

43          Side-effect function:   write_into_archive ( wrap_up )
```

Fig. 11

```
1   Module:  get_recent_contacts_for_this_customer

2      Submodule of:  info_about_customer

3      Input attributes:     account_number

4      Output attributes:    recent_contacts : list ( tuple ( date: date,
5                                                     event: event_type,
6                                                     delay_during_contact: int,
7                                                          \\ minutes
8                                                     delay_before_shipment: int
9                                                          \\ days
10                                                    amount: $value ) )
11     Enabling condition:   true

12     Type:                 foreign

13     Computation:          using recent_contacts_db
14                           select date,event,amount
15                           from contact_db
16                           where acct_num = account_number

17     Side-effect:          no
```

Fig. 12

```
1   Module:  get_recent_purchases_for_this_customer

2      Submodule of:  info_about_customer

3      Input attributes:     account_number

4      Output attributes:    recent_purchases : list ( tuple ( date: date,
5                                                        item : 20digit,
6                                                        qty : int,
7                                                        amount : $value ) )
8      Enabling condition:   true

9      Type:           foreign

10     Computation:          using purchase_db
11                           select date,item,qty,amount
12                           from purchases
13                           where acct_num = account_number

14     Side-effect:          no
```

Fig. 13

```
1   Module:  get_account_history_for_this_customer

2      Submodule of:  info_about_customer

3      Input attributes:     account_number

4      Output attributes:    account_history : tuple ( overdue_amount:
5                                                          $value,
6                                                    number_days_overdue:
7                                                          int,
8                                                    history: list ( tuple (
9                                                          date: date,
10                                                   item : 20digit,
11                                                   amount : $value )))
12     Enabling condition:   true

13     Type:           foreign

14     Computation:          using account_history_db
15                           select over_amt, num_days,history
16                           from account_history
17                           where acct_num = account_number

18     Side-effect:          no
```

Fig. 14

```
Module: calculate_frustration_score

   Submodule of:  info_about_customer

   Input attributes:     recent_contacts

   Output attributes:    frustration_score : [1..10]

   Enabling condition:   VAL(recent_contacts)

   Type:           decision

   Computation:
      Rules:             if recent_contacts#1 defined then
                         frustration_score <-
                                    (value/50) *
                                    [(delay_during_contact/2) +
                                    max(0,delay_before_shipment -
                                    10)/3]

                         if recent_contacts#2 defined then
                         frustration_score <-
                                    (value/100) *
                                    [(delay_during_contact/2) +
                                    max(0,delay_before_shipment -
                                    10)/3]

      Combining policy: frustration-score-cp //add contributions
                                               of true rules and
                                               round up, to max
                                               of 10

   Side-effect:          no
```

Fig. 15

```
1   Module:  calculate_net_profit_score

2      Submodule of:  info_about_customer

3      Input attributes:     recent_contacts,
4                            recent_purchases,
5                            account_history,
6                            cust_rec

7      Output attributes:    net_profit_score

8      Enabling condition:   recent_purchases#1.date<=now-60

9      Type:           decision

10     Computation:
11        Rules:                    if recent_purchases not empty then
12                                  net_profit_score <-
13                                  10% * sum (recent_purchases#i.amount
14                                  where recent_purchases#i.date > now -
15                                  60)

16                                  if recent_contacts not empty then
17                                  net_profit_score <-
18                                  -( 5 * count ( recent_contacts#i
19                                  where recent_contacts#i.type =
20                                  "complaint"))

21                                  if account_history.overdue_amount > 0
22                                  then net_profit_score <-
23                                  - account_history.overdue_amount *
24                                  account_history.number_days_overdue / 30

25                                  if cust_rec.card_color = "platinum" then
26                                  net_profit_score <- 100

27                                  if cust_rec.card_color = "gold" then
28                                  net_profit_score <- 50

29                                  if cust_rec.card_color = "green" then
30                                  net_profit_score <- 10

31                                  if DISABLED(cust_rec) then
32                                  net_profit_score <- 20

33          Combining policy:       net-profit-score-cp //add contributions
34                                                      of rules with true
35                                                      conditions
36
37      Side-effect:          no
```

Fig. 16

```
1   Module:  calculate_late_payment_score

2      Submodule of:  info_about_customer

3      Input attributes:     account_history

4      Output attributes:    late_payment_score

5      Enabling condition:   VAL(account_history)

6      Type:           decision

7      Computation:
8         Rules:             if cust_rec.card_color = "platinum" then
9                            late_payments_score <-
10                           (account_history.overdue_amount
11                           number_of_days_overdue)/100

12                           if cust_rec.card_color = "gold" then
13                           late_payments_score <-
14                           (account_history.overdue_amount *
15                           number_of_days_overdue)/50

16                           if cust_rec.card_color = "green" then
17                           late_payments_score <-
18                           (account_history.overdue_amount *
19                           number_of_days_overdue)/10

20     Combining policy:     late-payment-score-cp //rule with true
21                                                   condition wins;
22                                                   default is 0
23
24     Side-effect:          no
```

Fig. 17

```
Module:  calculate_cust_value

   Submodule of:  info_about_customer

   Input attributes:     net_profit_score,
                         late_payments_score,
                         cust_rec

   Output attributes:    cust_value

   Enabling condition:   true

   Type:           decision

   Computation:
      Rules:             if VAL(net_profit_score) then cust_value <-
                                    (1 - 1/net_profit_score) * 75

                         if cust_rec.card_color = "platinum" then
                         cust_value <- 20

                         if cust_rec.card_color = "gold" then cust_value
                         <- 10

                         if cust_rec.card_color = "green" then
                         cust_value <- 5

                         if VAL(frustration_score) then cust_value ←
                         5*frustration_score

      Combining policy: calculate-cust-val-cp //Add values of true
                                                rules and round up, to
                                                max of 100, default is
                                                0

   Side-effect:          no
```

Fig. 18

```
1   Module:  calculate_marketing_vs_collections

2      Submodule of:  info_about_customer

3      Input attributes:     cust_value,
4                            late_payments_score

5      Output attributes:    marketing_vs_collections

6      Enabling condition:   late_payments_score > 0

7      Type:           decision

8      Computation:
9         Rules:             if late_payments_score > f(cust_value) then
10                           marketing_vs_collections <- "collect"
11                           // f is function from [1..100] into [1..10],
12                           // it could be linear, i.e., f(cust_value) =
13                           //  cust_value/10
14                           // or it could be shallower in beginning and
15                                 steeper
16                           // towards end
17
18
19        Combining policy :      marketing-vs-collection-cp //default is
20                                                             "marketing",
21                                                             any rule
22                                                             with true
23                                                             condition
24                                                             wins
25
26     Side-effect:          no
```

Fig. 19

```
1   Module: Ask_Reason_For_Call

2       Submodule of:  routing_decisions

3       Input attributes:     < none >

4       Output attributes:    IVR_choice

5       Enabling condition:   cust_value < 7 and DNIS not =
6                             "Australia_promotion"

7       Type:           foreign

8       Computation:          x := IVR_dip( question(2));
9                             if x = 1 then IVR_choice := "dom";
10                            else if x = 2 then IVR_choice := "intl";
11                            else IVR_choice[state] = EXC and
12                            IVR_choice[EXC]=1
13
14          Side-effect:            yes

15          side-effect-function: IVR_dip( question(2))
```

Fig. 20

```
1   Module: calculate_business_value_of_call

2       Submodule of:  routing_decisions

3       Input attributes:     IVR_choice,
4                             web_destinations,
5                             frustration_score,
6                             marketing_vs_collections,
7                             late_payments_score,
8                             net_profit_score

9       Output attributes:    business_value_of_call : int

10      Enabling condition:   true

11      Type:           decision

12      Computation:
13         Rules:
14                       if true then business_value_of_call <-
15                             (cust_value/50 * net_profit_score)

16                       if true then business_value_of_call <-
17                             10*frustration_score

18                       if DNIS = "Australia_promotion" then
19                             business_value_of_call <- 100

20                       if "Australia" in web_destinations[i].regions for
21                             some i where
22                             web_destinations[i].date_last_modified > now -
23                             30
24                             then business_value_of_call <- 100

25                       if IVR_choice = "intl" then business_value_of_call <-
26                             50

27                       if marketing_vs_collections = "collect" then
28                             business_value_of_call <-
29                             (late_payments_score *
30                             account_history.overdue_amount)/5

31  Combining policy: business-value-of-call-cp // Add contributions of
32                                               rules with true
33                                               conditions and round up,
34                                               default is 0
35
36      Side-effect:          no
```

Fig. 21

```
1   Module: Calculate_send_bonus_check

2      Submodule of:  routing_decisions

3      Input attributes:     cust_rec

4      Output attributes:    send_bonus_check?

5      Enabling condition:   if net_profit_score > 1000
6                            and cust_rec.last_sent_bonus_check < now - 60
7                            and marketing_vs_collections = "market"
8                            OR
9                            if net_profit_score > 500
10                            and frustration_score > 8
11                           and cust_rec.last_sent_bonus_check < now - 60
12                            and marketing_vs_collections = "market"
13

14    Type:              foreign

15    Side-effect:             yes

16         side-effect-function:

17              issue_and_send_check($50,cust_rec.name,cust_rec.address)
```

Fig. 22

```
1   Module: call_priority

2       Submodule of:   routing_decisions

3       Input attributes:     business_value_of_call
4                             frustration_score

5       Output attributes:    call_priority

6       Enabling condition:   true

7       Type:           decision

8       Computation:
9          Rules:             if business_value_of_call < 25 then
10                                  call_priority <- 1

11                            if 25 =< business_value_of_call < 100  then
12                                  call_priority <- 2

13                            if 100 =< business_value_of_call < 500 then
14                                  call_priority <- 3

15                            if 500 =< business_value_of_call then
16                                  call_priority <- 4

17                            if frustration_score > 8 then
18                                  call_priority <- 4

19                            if 6 =< frustration_score <= 8 then
20                                  call_priority <- 3

21           Combining policy: call-priority-cp // high value wins with
22                                                 default result 2
23
24       Side-effect:          no
```

Fig. 23

```
Module: calculate_skill

    Submodule of:     routing_decisions

    Input attributes:       business_value_of_call
                            marketing_vs_collections
                            IVR_choice
                            DNIS
                            web_destinations

    Output attributes:      skill

    Enabling condition:     true

    Type:             decision

    Computation:
        Rules:        if marketing_vs_collections = "collections"
                            then skill <- ["collections", infinity]

                      if business_value_of_call > 100
                            then skill <- ["high_tier", 40]

                      if DNIS = "australia_promotion" then
                            skill <- ["australia_promo", infinity]

                      if "Australia" in web_destinations[i].regions
                            for some i where web_destinations[i].date_last_modified >
                            now - 30 then
                            skill <- ["australia_promo", 20]

                      if cust_rec.estimated_income_bracket = ">100K-150K" then
                            skill <- ["australia_promo", 25]

                      if cust_rec.estimated_income_bracket = ">150K" then
                            skill <- ["australia_promo", 35]

                      if IVR_choice = "dom" then skill <- ["norm_tier_dom",30]

                      if IVR_choice = "intl" then skill <- ["norm_tier_intl",30]

                      if "US" in web_destinations[i].regions for some
                            i where web_destinations[i].date_last_modified >
                            now - 30 then
                            skill <- ["norm_tier_dom", 20]

                      if "US" not in web_destinations[i].regions for
                            some i where web_destinations[i].date_last_modified > now -
                            30 then
                            skill <- ["norm_tier_intl", 20]

        Combining policy: skill-cp //weighted sum policy, and ties are
                                   broken by ordering "collections",
                                   "australia_promo", "high_tier",
                                    "low_tier_intl", "low_tier_dom",
                                    default is ⊥

    Side-effect: no
```

Fig. 24

```
Module: calculate_on_queue_promo
   Submodule of:  routing_decisions
   Input attributes:    promo_hit_list
   Output attributes:   on_queue_promo
   Enabling condition:  DISABLE if business_value > 100 or
frustration_score > 5
   Type:          decision
   Computation:
      Rules:            if 60 < ACD.expected_wait_time(skill)
                              then on_queue_promo <-
                              promo_hit_list[#1]

                        if business_value_of_call < 30
                              then on_queue_promo <- promo_hit_list[#1]

      Combining policy: on-queue-promo-cp // first true wins, default
                                             is 0

   Side-effect:         no
```

Fig. 25

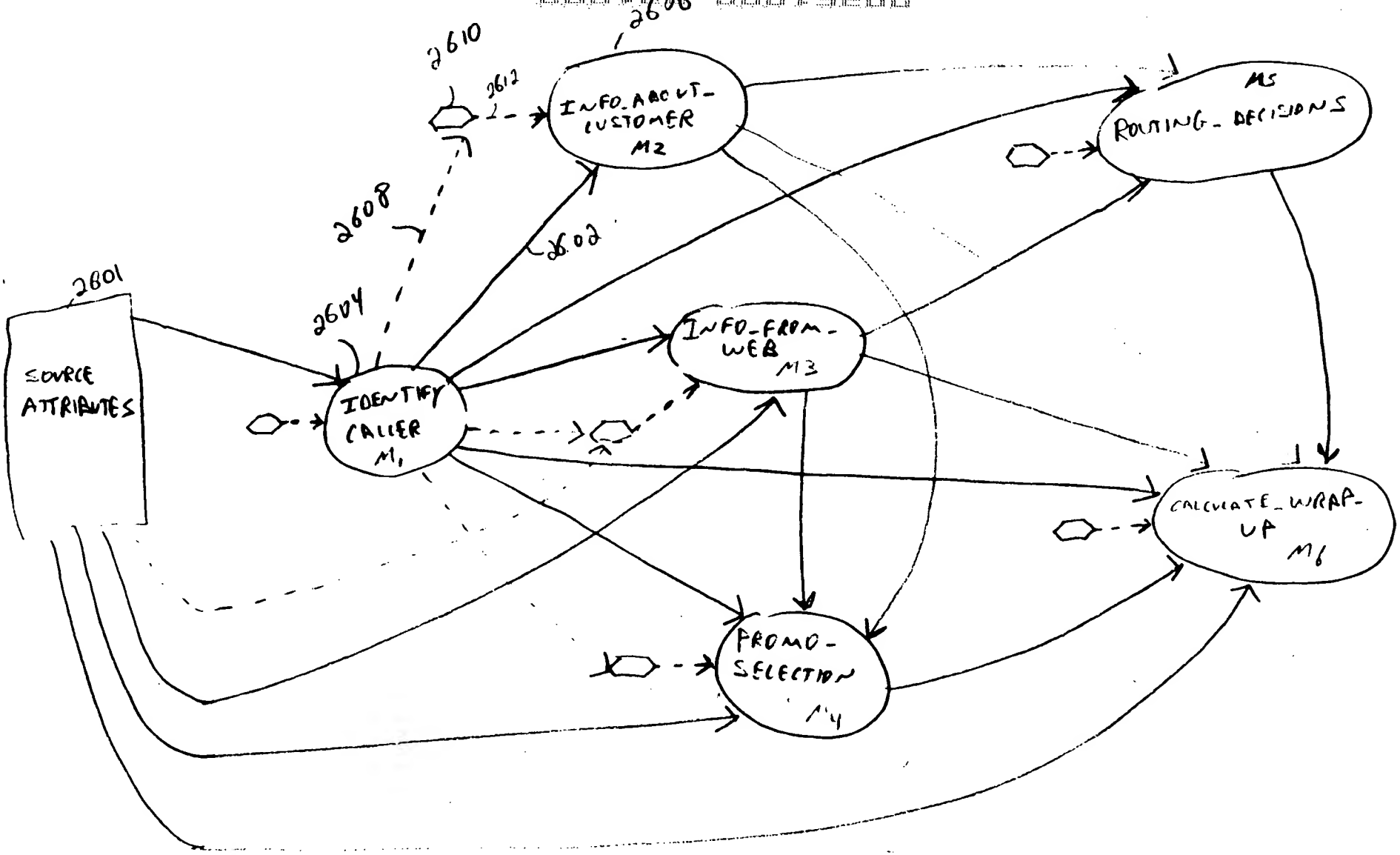
2606
2610
2612
INFO_ABOUT_CUSTOMER
M2
M5
ROUTING_DECISIONS
2608
2602
2601
2604
SOURCE ATTRIBUTES
IDENTIFY CALLER
M1
INFO_FROM_WEB
M3
CALCULATE_WRAP-UP
M6
PROMO_SELECTION
M4

FIG 26

$$\frac{\sigma\vdash e:t}{\sigma\vdash \mathsf{value}(e):bool}$$ value

$$\frac{\sigma\vdash f:AM:t_1\times\cdots\times t_n\to t,\sigma\vdash e_1:t_1,\cdots\sigma\vdash e_n:t_n}{\sigma\vdash \mathsf{Apply}((f,e_1,\cdots,e_n)):t}$$ apply

$$\frac{\sigma\vdash e_1:t_1,\cdots\sigma\vdash e_n:t_n}{\sigma\vdash (e_1,\cdots,e_n):(a_1:t_1,\cdots,a_n:t_n)}$$ tupling

$$\frac{\sigma\vdash e_1:t,\cdots,\sigma\vdash e_n:t}{\sigma\vdash \{e_1,\cdots,e_n\}:\{t\}}$$ bagging

$$\frac{\sigma\vdash e_1:t,\cdots,\sigma\vdash e_n:t}{\sigma\vdash [e_1,\cdots,e_n]:[t]}$$ listing

$$\frac{\sigma\vdash e:\{t\}}{\sigma\vdash \mathsf{unitval}(e):t}$$ unitval

$$\frac{\sigma\vdash e:(a_1:t_1,\cdots,a_n:t_n)}{\sigma\vdash e.a_i:t_i}$$ projection on tuples

$$\frac{\sigma\vdash e:[t]}{\sigma\vdash e\#i:t}$$ projection on lists

$$\frac{\sigma\vdash e_1:[t_1],\sigma\vdash e_2:t_2}{\sigma\vdash \mathsf{factor}(e_1,e_2):[(f\_a:t_1,s\_a:t_2)]}$$ factor (on lists)

$$\frac{\sigma\vdash e_1:\{t_1\},\sigma\vdash e_2:t_2}{\sigma\vdash \mathsf{factor}(e_1,e_2):\{(f\_a:t_1,s\_a:t_2)\}}$$ factor (on bags)

$$\frac{\sigma\vdash f:t_1\to t,\sigma\vdash S:[t_1]}{\sigma\vdash \mathsf{map}(f)(S):[t]}$$ map (on lists)

$$\frac{\sigma\vdash f:t_1\to t,\sigma\vdash S:\{t_1\}}{\sigma\vdash \mathsf{map}(f)(S):\{t\}}$$ map (on bags)

$$\frac{\sigma\vdash id_\theta:t,\sigma\vdash\theta:t\times t\to t,\sigma\vdash S:\{t\}}{\sigma\vdash \mathsf{collect}(id_\theta,\theta)(S):t}$$ collect (on bags)

$$\frac{\sigma\vdash id_\theta:t,\sigma\vdash\theta:t\times t\to t,\sigma\vdash S:[t]}{\sigma\vdash \mathsf{collect}(id_\theta,\theta)(S):t}$$ collect (on lists)

FIG. 27

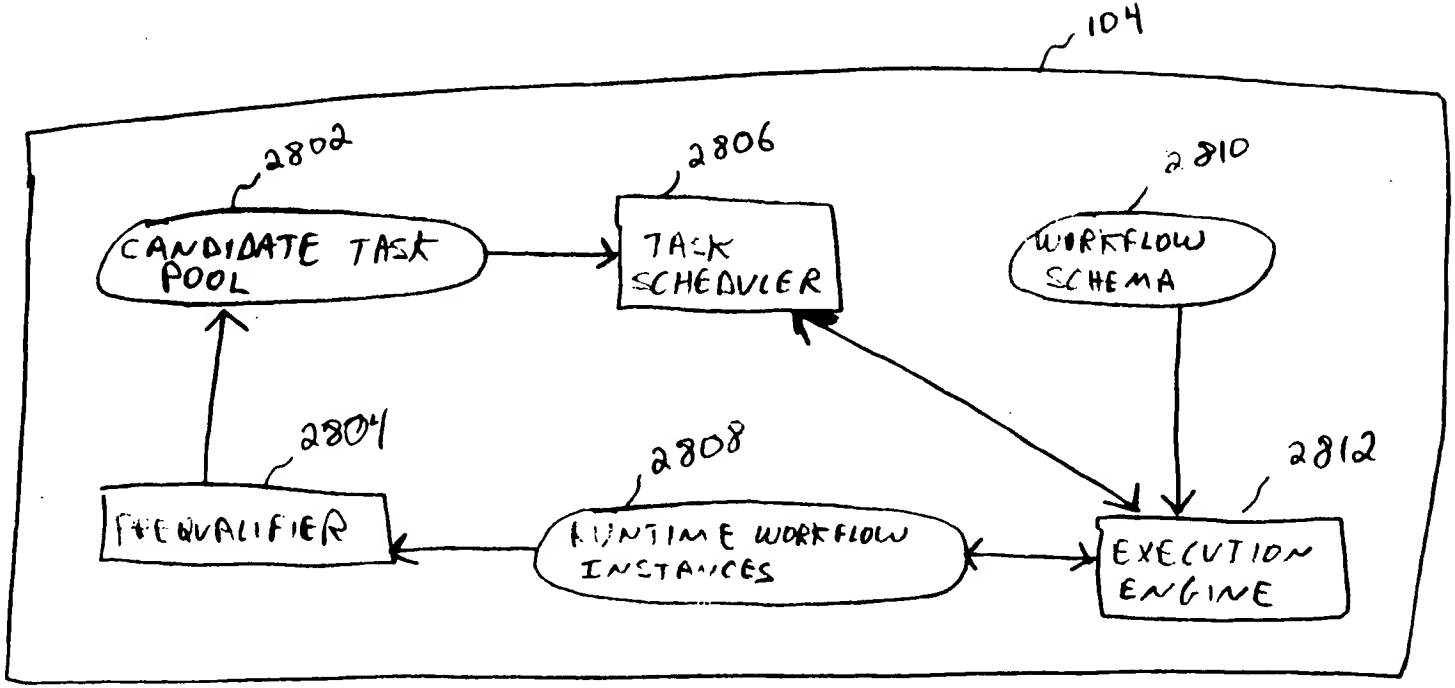
104
2802
CANDIDATE TASK POOL
2806
TASK SCHEDULER
2810
WORKFLOW SCHEMA
2804
PREQUALIFIER
2808
RUNTIME WORKFLOW INSTANCES
2812
EXECUTION ENGINE

FIG. 28

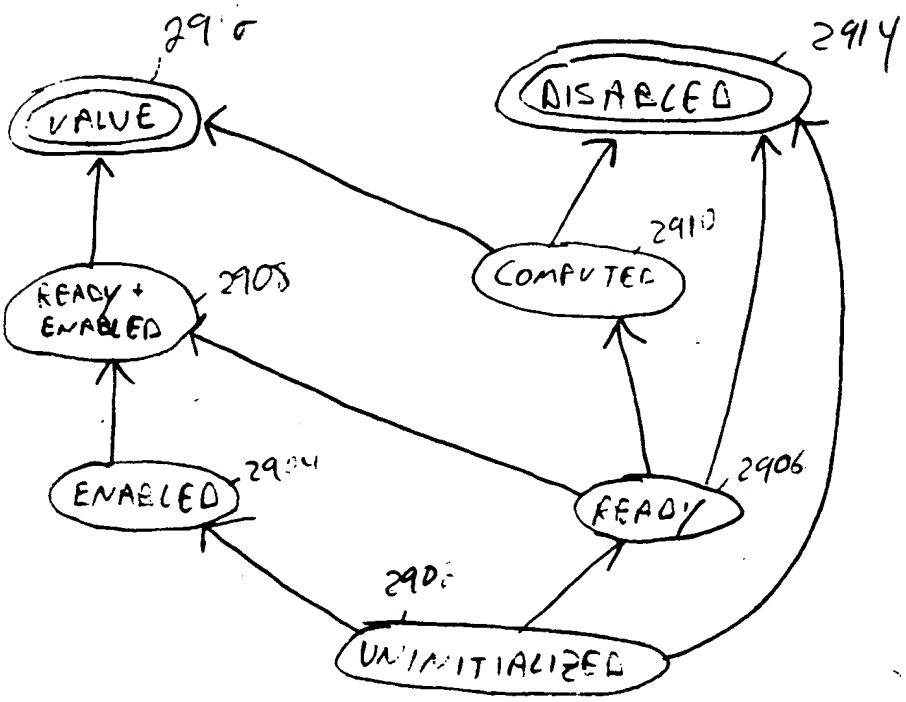
2916
VALUE
2914
DISABLED
2910
COMPUTED
READY + ENABLED
2908
ENABLED
2904
READY
2906
2902
UNINITIALIZED

FIG 29

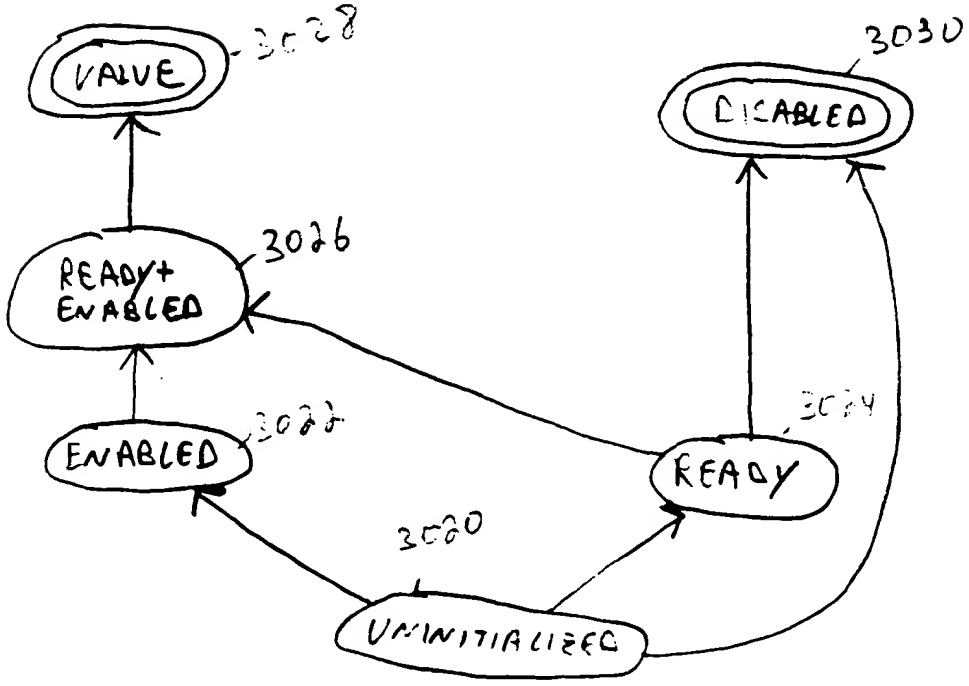
3028
VALUE
3030
DISABLED
READY+ ENABLED
3026
ENABLED
3022
3024
READY
3020
UNINITIALIZED

FIG 30

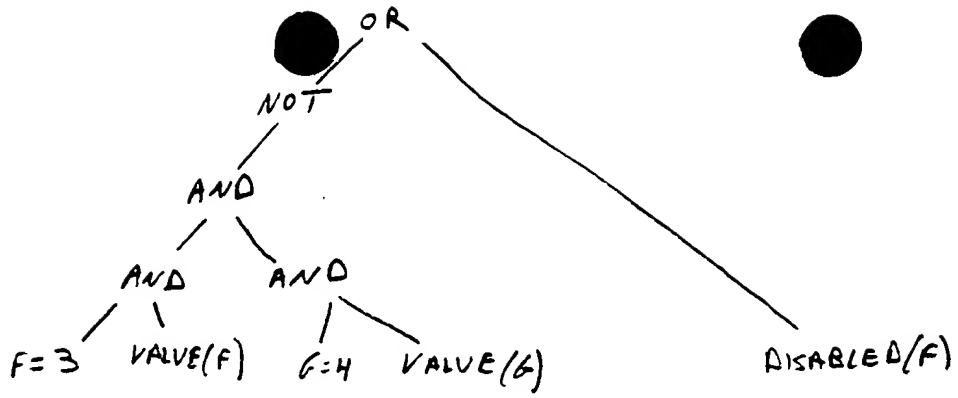

FIG. 31

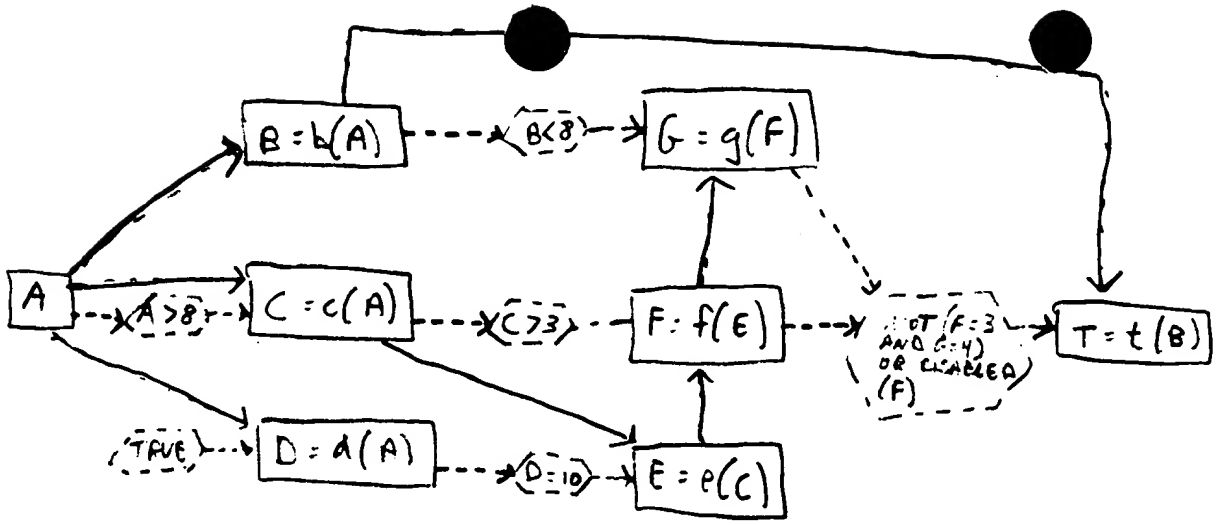
A
B = b(A)
C = c(A)
D = d(A)
E = e(C)
F = f(E)
G = g(F)
T = t(B)
A > 8
C > 3
TRUE
D = 10
B < 8
NOT (F = 3 AND G = 4) OR CLEARED (F)

FIG. 32

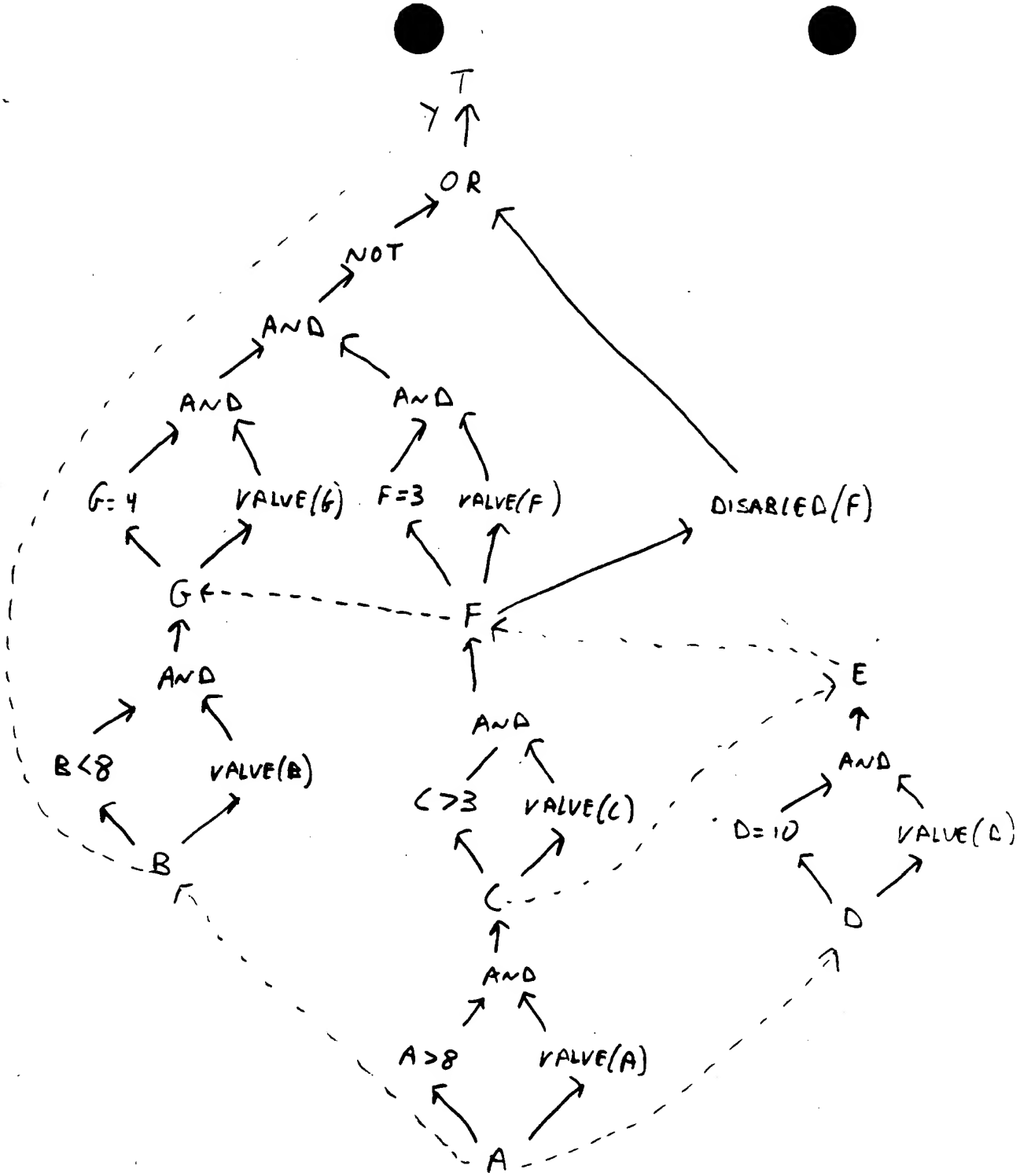
T
OR
NOT
AND
AND
AND
G=4
VALVE(G)
F=3
VALVE(F)
DISABLED(F)
G
F
AND
AND
E
B<8
VALVE(B)
C>3
VALVE(C)
AND
D=10
VALVE(D)
B
C
D
AND
A>8
VALVE(A)
A

FIG. 33

Global variables:

These variables are global to the whole execution of workflow instance
- G: a dependency graph
- S: set of source attribute nodes of G
- T: set of target attribute nodes of G
- σ []: array of attribute states
- μ []: array of attribute values
- α []: array of three valued logic values (*true, false unknown)*
- *HIDDEN_EDGE:* set of hidden edges of G.
- *HIDDEN_ATT:* set of hidden attribute nodes of G.

3402

Notations:

- $\sigma[A]$: element of array σ [] that corresponds to the attribute node A in G
- $\mu[A]$: element of array μ [] that corresponds to the attribute node A in G
- $\alpha[p]$: element of array α [] that corresponds to the condition node p in G

3404

Initialization phase:

procedure Init:

Input:

g: a dependency graph:
So: source nodes in g
Te: terminal nodes in g

body:

```
BEGIN init
    G:=g ; S : = So: T : = Te;
    /*Initialization of the states and values of attributes nodes */
    FOR all the attribute nodes A in G DO
       IF A ∈ S    /* A is a source node */
         THEN σ [A] := READY + ENABLED
         ELSE σ [A] := UNITIALIZED;
      μ [A] := NULL;
    END FOR
    /* Initialization of α-values of condition nodes */
    FOR all the condition nodes p in G DO
      α[A] := unknown;
    END FOR
    */ Initialization of the set of hidden edges and hidden nodes */
     HIDDEN_EDGE := ø; HIDDEN_ATT := ø
END init
```

3408

3410

3412

3406

FIG 34A

Increment

Input:

A : an attribute in *G*.

v : a value for *A*.

body:

BEGIN increment

$\mu[A] := v$;

IF $\sigma[A]$ = READY

THEN propagate_att_change(*A*, COMPUTED)

IF $\sigma[A]$ = READY+ENABLED

THEN propagate_att_change(*A*, VALUE)

END Increment

propagate_att_change

Input:

B : an attribute in *G*.

σ : a state for *B*

body:

/* Set state for *B*1

IF ((σ[*B*] = ENABLED) AND (σ = READY)) OR (σ [*B*] = READY) AND (σ =ENABLED))

THEN σ [*B*] := READY+ENABLED

ELSE σ [*B*] := σ;

/* push relevant information to the affected successor nodes */

CASE : σ [*B*] ∈ {VALUE, COMPUTED} /* The value of *B* is computed */

/* try to evaluate predicate nodes that are using the value of *B* */

FOR each condition node *p* of the form *pred*(t_1, ..., t_n) *such that* (*B*,*p*) ∈ *G* DO

IF (*B*,*p*) ∉ *HIDDEN_EDGE*

THEN

Hide_edge ((*B*,*p*));

IF *Eval* (*p*) ≠ *unknown* THEN α [*p*] := *Eval*(*p*); propagate_cond_change(*p*)

END FOR

/* check if the attributes nodes that have B as input parameters are READY */

FOR each attribute node C such that (*B*, *C*) ∈ *G* DO

IF σ[*B*]=VALUE THEN

IF (*B*, *C*) ∉ *HIDDEN_EDGE*

THEN

Hide_edge((*B*,*C*));

IF there exists no attribute node *D* such that (*D*, *C*) ∉ *HIDDEN_EDGE*

THEN propagate_att_change (*C*, READY);

END FOR

CASE : σ [*B*] = ENABLED

/* evaluates condition nodes of the form VALUE (*B*) and DISABLED (*B*) */

FOR each condition node *p* of the form VALUE (*B*) or DISABLED (*B*) such that (*B*,*p*) ∈ *G* DO

IF (*B*,*p*) ∉ *HIDDEN_EDGE*

FIG 34B

```
            THEN
              Hide_edge((B,p))
              IF p is of the form VALUE (A) THEN α[p] := true ELSE α[p]: = false;
              propagate_cond_change(p);
      END FOR
CASE : σ [B] = DISABLED
    /* evaluate condition nodes of the form VALUE (B) and DISABLED (B) */
    FOR each condition node p of the form VALUE (B) or DISABLED (B) such that (B,p) ∈ G DO
      IF (B,p) ∉ HIDDEN_EDGE
         THEN
           Hide_edge ((B,p));
           IF p is of the form VALUE (A) THEN α[p] := false ELSE α[p] := true;
           propagate_cond_change(p);
    END FOR
    /* check if the attribute nodes that have B as input parameters are READY */
    FOR each attribute node C such that (B,C) ∈ G DO
      IF (B,C) ∉ HIDDEN_EDGE
           THEN
             Hide_edge((B,C));
             IF there are no more attribute nodes D such that (D,C) ∉ HIDDEN_EDGE
               THEN propagate_att_change (C,READY);
    END FOR
    /* If the attribute is stable then hide the attribute */
    IF (σ [B] ∈ {DISABLED, VALUE}) THEN Hide_node(B);
    END propagate_att_change

    propagate_cond_change
          Input:
            p: a condition node in G.
          body:
          BEGIN propagate_cond_change
          let n be the successor of p in G
          IF (p,n) ∉ HIDDEN_EDGE
            THEN
              Hide _edge ((p,n));
              CASE: n is OR condition node
                IF (α [p] = true) THEN α [n] : = true; propagate_cond_change(n); END IF;
                IF α [p] = false AND for each condition node p' where (p',n) ∈ G, (p',n) ∈
                HIDDEN_EDGE
                    THEN α [n] : = false; propagate_cond_change(n);END IF;
              CASE: n is a AND node
                IF (α [p] = false) THEN α [n] := false; propagate_cond_change(n);END IF;
                IF α [p] = TRUE AND for each condition node p' where (p',n) ∈ G, (p',n) ∈
                HIDDEN_EDGE
```

3442 3444 3446 3448 3452 3450 3454 3456 3458 3460 3462 3466 3468 3464

FIG. 34C

```
        THEN α [n]: = TRUE ; propagate_cond_change(n);END IF;
    CASE : n is NOT node
        α [n]: = ¬ (α [p]) ; propagate_cond_change(n);
    CASE : n is an attribute node
        IF (α [p] = true)
            THEN propagate_att_change(n, ENABLED)
            ELSE propagate_att_change(n,DISABLED);
END propagate_cond_change
```

```
Hide_edge
 Input
    (n,n') : an edge in G.
body
BEGIN Hide_edge
HIDDEN_EDGE : = HIDDEN_EDGE ∪ {(n,n')];
IF (there are no more edges (n, n") ∈ G such that (n,n") ∉ HIDDEN_EDGE
    THEN Hide_node(n)
END Hide_edge
```

```
Hide_node
    Input
        n : a node in g.
    body
    BEGIN Hide_node
    HIDDEN_ATT := HIDDEN_ATT ∪ {n}
    FOR each edge (n',n) ∈ g such that (n',n) ∉ HIDDEN_EDGE) DO
        Hide_edge (n',n)
     END FOR
  END Hide_node
```

FIG 34 D

Global variables:

These variables are global to the whole execution of workflow instance

G : a dependency graph
S : set of attribute nodes of G /* this set contains the source nodes */
T : set of attribute nodes of G /* this set contains target nodes */
$\sigma[]$: array of attribute states
$\alpha[]$: array of three valued logic values (*true, false unknown*)
HIDDEN_EDGE : set of edges of G.
HIDDEN_ATT : set of attribute nodes of G.

$T\_N[][]$: Matrix of integers that associates an integer value to each pair (p,A) where p is a condition node and A is an attribute node in G
/* $T\_N[p][A] = 0$ means that the attribute A is True_necessary for the the condition node p*/

$F\_N[][]$: Matrix of integers that associates an integer value to each pair $(p;A)$ where p is a condition node and A is an attribute node in G
/*$F\_N[p][A] = 0$ means that the attribute A is False_necessary for the condition node p*/

$V\_N[][]$: Matrix of integers associates an integer value to each pair (B,A) where B and A are attribute nodes in G
/*$V\_N[B][A] = 0$ means that the attribute A is Value_necessary for the attribute node B*/

$S\_N[][]$: Matrix of integers associates an integer value to each pair (B,A) where B and A are attribute nodes in G
/*$S\_N[B][A] = 0$ means that the attribute A is Stable_necessary for the attribute node B*/

$N[]$: Array of boolean
$N[A]$ = *true* means that the attribute A is computed as necessary/*
$N[A]$ = *false* means that the attribute A is not computed as necessary*/

Notations :

nb_pred(p) : number of predecessors of p in G

Initialization phase:
procedure Init :
Input:
g : a dependency graph:
So : source nodes in g
Te : terminal nodes in g
body:
BEGIN N_init

Init() } 3508

```
/* Initialization of T_N,F_N,S_N,V_N */
FOR all the condition nodes p in G DO
   FOR all the attribute nodes A in G DO

      CASE : p is an OR node :
         T_N[p][A] := nb_pred(p):                    /* rule 1 */
         F_N[p][A] := 1;                             /* rule 2 */

      CASE : p is an AND node :
         T_N[p][A] := 1;                             /* rule 3 */
         F_N[p][A] := nb_pred(p);                    /* rule 4 */

      CASE : p is a NOT node :
         T_N[p][A] := 1;                             /* rule 5 */
         F_N[p][A] := 1;                             /* rule 6 */

      CASE : p is a node of the form VAL(B) or DIS(B):
         T_N[p][A] := 1;                             /* rules 7 and 9 */
         F_N[p][A] := 1;                             /* rules 8 and 10 */

      CASE : p is a node of the form pred(t1,...tn):
         T_N[p][A] := 1;                             /* rule 11 */
         F_N[p][A] := 1                              /* rule 12 */

   END FOR
END FOR

FOR all the attributes nodes A in G DO
   FOR all the attribute nodes B in G DO
      S_N[A][B] := 1; V_N[A][B] := 1
   END FOR
END FOR

FOR all the attributes nodes A in G DO
   N[A] := false
END FOR

END N_init
```

3511 (rule 1 / rule 2)
3510 (CASE p is an AND node)
3512 (FOR all the attributes nodes A in G DO)
3513 (FOR all the attributes nodes A in G DO)
3506 (END N_init)

N_Increment

Input :

A : an attribute in *G*.

v : a value for *A*.

Variables /* Global to one execution of the increment phase (for one execution step) */ 3518

FIG 35B

prev_E: set of attribute nodes
/* used to store the nodes that were READY+ENABLED or ENABLED (in a previous execution of N-increment) */

prev_HIDDEN_EDGE: /* set of edges*/
used to store the edges that were previously hidden (in the previous steps) */

prev_T_N: set of pairs (p,A) where p is a condition node and A is an attribute node
/* used to denote the elements of $T\_N[p][A]$ that were set to 0 in a previous execution of N-increment*/

prev_F_N: set of pairs (p,A) where p is a condition node and A is an attribute node
/* used to denote the elements of $F\_N[p][A]$ that were set to 0 in a previous execution of N-increment*/

3518

Δ_E: set of attribute nodes
/* used to store the new ENABLED or READY+ENABLED attribute nodes that were neither ENABLED nor READY+ENABLED in the previous steps. */
Δ_*HIDDEN_EDGE* : set of edges
/* used to store the new hidden edges */

3520

new_V_N : set of pair (A,A) where A is an attribute node
/* used to store the positions of elements of $V\_N[][]$ whose new value is zero due to case 1 */
new_S_N : set of pair (B,A) where B and A are attribute nodes
/* used to store the positions of elements of $S\_N[][]$ whose new value is zero due to case 2 */
new_T_N : set of pair (p,A) where p is a predicate node and A is an attribute node
/* used to store the positions of elements of $T\_N[][]$ whose new value is zero due to some new hidden edges (case 3) */
new_F_N : set of pair (p,A) where p is a predicate node and A is an attribute node
/* used to store the positions of elements of $F\_N[][]$ whose new value is zero due to some new hidden edges (case 4) */

3522

351

body:
BEGIN N_increment

/* preparation step: */
prev_HIDDEN_EDGE := *HIDDEN_EDGE*;
*prev*_E := {A | A is an attribute node in G and σ [A] $\in$ {READY+ENABLED, ENABLED}

3524

Increment(A, v)

3526

/* Instigation step : Compute new necessary properties according to the instigation cases*/

3528

Fig 35C

Case 1 :

$\Delta\_E := \{A | A$ is an attribute node in G and $\sigma[A] \in$ {READY+ENABLED, ENABLED} and $A \notin prev\_E\}$

$new\_V\_N := \emptyset$;

FOR each attribute node A in Δ_E DO

$V\_N[A][A] := 0$; $new\_V\_N := new\_V\_N \cup \{(A,A)\}$/* a node is value_necessary for itself*/

END FOR

3530

Case 2 :

$new\_S\_N := \emptyset$;

FOR each attribute node B in Δ_E DO

FOR each attribute node in A in G such that $\sigma [A] \in$ {READY+ENABLED, ENABLED} DO

IF $V\_N[B][A] = 0$ and $S\_N[B][A] = 1$

THEN $S\_N[B][A] = 0$; $new\_S\_N := new\_S\_N \cup \{(B,A)\}$ /* **rule (13)***/

END FOR

END FOR

3532

$\Delta\_HIDDEN\_EDGE := HIDDEN\_EDGE - prev\_HIDDEN\_EDGE$

$prev\_T\_N := \{(p,A) \mid T\_N[p][A] = 0\}$

$prev\_F\_N := \{(p,A) \mid F\_N[p][A] = 0\}$

$new\_T\_N := \emptyset$;

$new\_F\_N := \emptyset$;

3534

FOR all edges $(n,p) \in \Delta\_HIDDEN\_EDGE$ such that $p \notin HIDDEN\_ATT$ and p is a condition node DO

FOR all attribute nodes A such that $\sigma (A) \notin$ {COMPUTED, VALUE, DISABLED} DO

CASE: 3

CASE : p is an OR node:

IF $(n,A) \notin prev\_T\_N$

THEN

$T\_N[p][A] := T\_N[p][A] - 1$; /* **rule (1)***/

IF $T\_N[p][A] = 0$ THEN $new\_T\_N := new\_T\_N \cup \{(p,A)\}$

3536

CASE: 4

CASE : p is an AND node :

IF (n,A) $\notin prev\_F\_N$ /* same reasoning as for OR nodes but with rule 4*/

THEN

$F\_N[p][A] := F\_N[p][A] - 1$; /* **rule (4)***/

IF $F\_N[p][A] = 0$ THEN $new\_F\_N := new\_F\_N \cup \{(p,A)\}$

END FOR

END FOR

3538

352

F/F 352

```
/* Propagation step */
   New_propagate(new_V_N, new_S_N, new_T_N, new_F_N)
END N_Increment

New_propagate
   Input :
    new_V_N : set of pairs (A,A) where A is an attribute node
    new_S_N : set of pairs (B,A) where B and A are attribute nodes
    new_T_N : set of pairs (p,A) where p is a condition node in G and A is an attribute
    node
    new_F_N : set of pairs (p,A) where p is a condition node in G and A is an attribute
    node
   body:
   FOR each pair (A,A) in new_V_N DO
      propagate_V_N(A,A)
      FOR each attribute node B such that (A,B) ∈ G and (A,B) ∉ HIDDEN_EDGE
        V_N[B][A] := 0; propagate_V_N(B,A)/*      rule (16) */
      END FOR
   END FOR
   FOR each pair (B,A) in new_S_N DO
       propagate_S_N(B,A)
   END FOR
     FOR each pair (p,A) in new_T_N DO
        propagate_T_N(p,A)
   END FOR
   FOR each pair (p,A) in new_F_N DO
        Propagate_F_N(p,A)
   END FOR
END N-propagate

propagate_V_N
   Input :
    B : an attribute node in G.
    A : an attribute node in G./* A is newly Value_necessary for B*/
   body:
   IF σ[B] = ENABLED and S_N[B][A] = 1
      THEN S_N[B][A] = 0; propagate_S_N(B,A)          /*rule (13) */
  ELSE let p be the condition node such that (p,B) ∈ G.
        IF F_N[p][A]=0 and S_N[B][A] = 1
          THEN S_N[B][A] = 0; propagate_S_N(B,A)          /*rule (14)*/
  END IF
  FOR each condition node p of the form pred(t1,...,tn)
        such that (B,p) ∈ g and (B,p) ∉ HIDDEN_EDGE DO
   IF T_N[p][A] = 1
      THEN T_N[p][A] : = 0;propagate_T_N(p,A)          /*rule (11)*/
   IF F_N[p][A] = 1
     THEN F_N[p][A] : = 0;propagate_F_N(p,A)          /*rule (12)*/
```

3540 3542 3544 3546 3548 3550 3552 3554 3556

FIG 35E

```
 END FOR
END propagate_V_N

propagate_S_N
  Input:
     B : an attribute node in G.
     A : an attribute node in G./* A is newly Stable_necessary for B*/
   body:
  FOR each attribute node C such that (B,C) ∈ g and (B,C) ∉ HIDDEN_EDGE DO
     IF V_M[C][A] = 1 THEN V_M[C][A] = 0;propagate_V_N(C,A)  /* Rule 17 */
  END FOR
  IF B ∈ T THEN N[A] : = true
END propagate_S_N

propagate_F_N
   Input:
      p : a condition node in G.
      A: an attribute node in G./* A is newly False_necessary for p*/
   body:
      let n be the successor of p in G
      IF (p,n) ∈ HIDDEN_EDGE
          THEN
            CASE : n is an OR or AND node
              IF F_M[n][A] > 0
                 THEN
                  F_M[n][A] := F_M[n][A] – 1;            */*rules (2) and (4)*/
                  IF F_M[n][A] = 0 THEN propagate F_N (n,A)
             CASE : n is a NOT node
               IF T_M[n][A] = 1 THEN T_M[n][A] : = 0;propagate_T_N(n,A) /*rule (6)*/
             CASE : n is an attribute node
               IF (T_M[p][A] = 0 or V_M[n][A] = 0 and S_M[n][A] = 1
                  THEN S_M[n][A] = 0;propagate_S_N(n,A)     /*rules (14) and (15)*/
                  FOR each condition node p' of the form VALUE (n)
                        such that (n,p') ∈ g and (n,p') ∉ HIDDEN_EDGE DO
           IF F_M[p'][A] = 1 THEN F_M[p'][A] : = 0;propagate_F_N(p',A) /*rule (8)*/
         END FOR
         FOR each condition node p' of the form DISABLED (n)
              such that (n,p') ∈ G AND (n,p') ∉ HIDDEN_EDGE DO
         IF T_M[p'][A] = 1 THEN (T_M[p'][A] := 0;propagate_T_N(p',A)  /*rule (10)*/
       END FOR
END propagate_F_N

propagate_T_N
   Input:
     p: a condition node in G.
     A: an attribute node in G/*  A is newly True_necessary for p*/
   body:
```

3542
3548
3558
3560
3562
3564
3566

FIG 25F

```
let n be the successor of p in G
IF (p,n) ∉ HIDDEN_EDGE
    THEN
     CASE : n is an OR or AND node
      IF T_M[n][A] > 0
          THEN
           T_M[n][A] := T_M[n][A] – 1;    /*rules (1) and (3)*/
           IF T_M[n][A] = 0 THEN propagate_T_N(n,A)
     CASE : n is a NOT node
      IF F_M[n][A] = 1 THEN F_M[n][A] : = 0; propagate_F_N(n,A) /* rule (5) */
     CASE : n is an attribute node
       IF F_M[p][A] = 0 and S_M[n][A] = 1
         THEN S_M[n][A] = 0;propagate_S_N(n,A)  /*rule (15)*/
        FOR each condition node p' of the form VALUE (n)
             such that (n,p') ∈ G and (n,p') ∉ HIDDEN_EDGE DO
         IF T_M[n][A] = 1 THEN
              T_M[p'][A] : = 0;propagate_T_N(p',A)       /*rule (8)*/
        END FOR
        FOR each condition node p' of the for DISABLED (n)
             Such that (n,p') ∈ G and (n,p') ∉ HIDDEN_EDGE DO
         IF F_M[n][A] = 1 THEN
          F_M[p'][A] : = 0;propagate_F_N(p',A)       /*rule (9)*/
        END FOR
END propagate_T_N
```

3566

3542

FIG 35

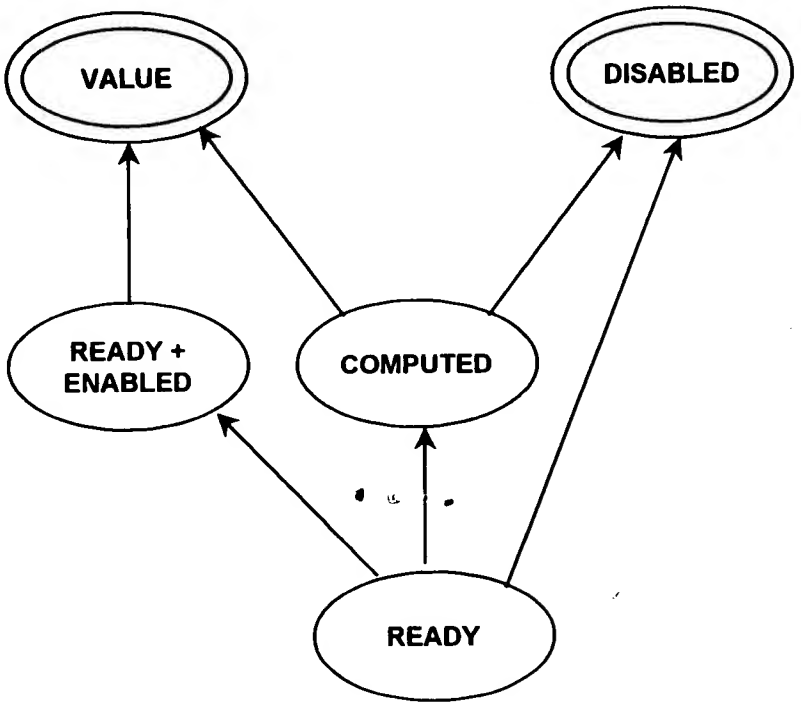

FIG 36

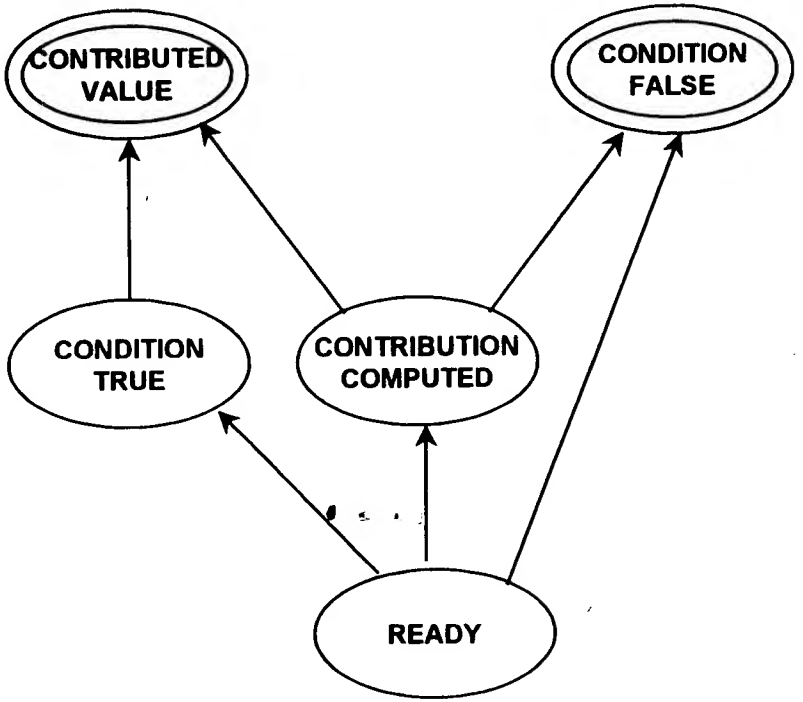

FIG. 37

	A	B	C	D	E	F	G	H	I	J
1	source		get_recent_contacts_... (node 504)	get_recent_purchases_... (node 508)	get_account_history_... (node 512)	calculate_frustration_score (node 516)	calculate_net_profit_score (node 520)	calculate_late_payments_score (node 524)	calculate_cust_value (node 528)	calculate_marketing_vs_collections (node 532)
2			foriegn module	foriegn module	foriegn module	"add contribs. of true rules and round up, to max of 10"	"add contribs. of true rules"	"true rule wins; default is 0"	"add contribs. of true rules and round up, to max of 100"	"any true rule gives collect; default is marketing"
3	cust_rec	account_number	recent_contacts	recent_purchases	account_history	frustration_score	net_profit_score	late_payment_score	cust_value	marketing_vs_collections
4	<"John Doe", "101 Ash, LA", "gold", FALSE ... >	421135	NS	NS	NS	NS	NS	NS	NS	NS
6			ENABLED, READY	ENABLED, READY	ENABLED, READY	READY	READY	READY	ENABLED, READY	READY
6						READY	READY	⊥	READY	"collect" C-C
7						READY	READY	CONDITION TRUE	⊥	
8							READY	⊥	10 C-V	
9							⊥		⊥	
10							50 C-V		READY	

FIG. 58

	A	B	C	D	E	F	G	H	I	J
1	source		get_recent_ contacts_... (node 504)	get_recent_ purchases_... (node 508)	get_account_ history_... (node 512)	calculate_ frustration_ score (node 516)	calculate_ net_profit_ score (node 520)	calculate_ late_payments_ score (node 524)	calculate_ cust_value (node 528)	calculate_ marketing_vs_ collections (node 532)
2			foriegn module	foriegn module	foriegn module	"add contribs. of true rules and round up, to max of 10"	"add contribs. of true rules"	"true rule wins; default is 0"	"add contribs. of true rules and round up, to max of 100"	"any true rule gives collect; default is marketing"
3	cust_rec	account_ number	recent_ contacts	recent_ purchases	account_ history	frustration_ score	net_profit_ score	late_ payment_ score	cust_ value	marketing_ vs_ collections
4	<"John Doe", "101 Ash, LA", "gold", FALSE SV ... >	421135 SV	NS	[<8-10-98,coat, 1, $50> <6-15-98,hat, SV 1, $20 >	<10,45,[<9-18-98 pay,$40 > <8-10-98, SV order,$50>]	NS	⊥ SU	9 SV	NS	NS
5			ENABLED & READY	VALUE	VALUE	READY	DISABLED	VALUE	ENABLED & READY	ENABLED & READY
6						READY	⊥	⊥	⊥	"collect" C-C
7						READY	READY	9 C-V	⊥	
8							-9 C-V	⊥	10 C-V	
9							⊥		⊥	
10							50 C-V		READY	

FIG 39

Initialization

```
Based on the DL specification, compute rows 1, 2, and 3 of the display;
For source attribute cells of row 4 do:
    For each source attribute with value, insert value and apply
"attribute_value_indication";
    For each source attribute that is disabled, apply
"attribute_disabled_indication";
For each non-decision module
    In row 5, apply "module_uninitialized_indication";
    In row 4, apply "attribute_uninitialized_indication";
For each decision module
    In row 5, apply "module_ready_indication";
    In row 4, apply "attribute_uninitialized_indication";
For each cell in rows 6,7,8,.., apply "rule_ready_indication"
```

Iteration

```
For each event of execution engine do
Case on event_type
    non_dec_module_enabled:
        in row 5, apply "module_enabled_indication";

    non_dec_module_ready:
        in row 5, apply "module_ready_indication";

    non_dec_module_ready+enabled:
        in row 5, apply "module_ready+enabled_indication";

    non_dec_module_computed::
        in row 5, apply "module_computed_indication";
        in row 4, label corresponding attribute cell with the value computed
    and apply
            "attribute_computed_indication";

    non_dec_module_value:
        in row 5, label cell for this module as "value" and apply
    "module_value_indication";
        in row 4, label corresponding attribute cell with value assigned and
    apply
            "attribute_value_indication"

    non_dec_module_disabled:
```

FIG 4A

in row 5, label cell for this module as "disabled" and apply
"module_disabled_indication";
in row 4, label corresponding attribute cell with "⊥" and apply
"attribute_disabled_indication"

dec_module_enabled+ready:
in row 5, label cell with "enabled+ready" and apply
"module_enabled+ready_indication";

dec_module_computed:
in row 5, label cell with "computed" and apply
"module_computed_indication";
in row 4, label cell with the computed value and apply
"attribute_computed_indication";

dec_module_value:
in row 5, label cell with "value" and apply
"module_value_indication";
in row 4, label cell with the computed value and apply
"attribute_value_indication";

dec_module_disabled:
in row 5, label cell with "disabled" and apply
"module_disabled_indication";
in row 4, label cell with "⊥" and apply
"attribute_disabled_indication";

comp_rule_condition_true:
to corresponding cell, apply "rule_cond_true_indication";

comp_rule_contribution_computed:
to corresponding cell, label with computed value and apply
"rule_contribution_computed_indication";

comp_rule_contributed_value:
to corresponding cell, label with computed value and apply
"rule_contributed_value_indication";

comp_rule_condition_false:
to corresponding cell, label with "⊥" and apply
"rule_condition_false_indication";

EndCase

4024
4026
4028
4030
4032
4034
4036
4038

FIG 40B